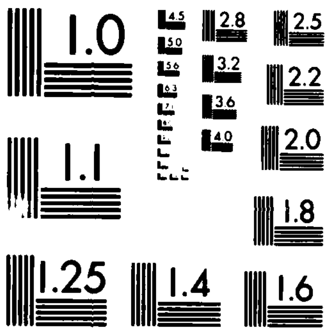

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

AD A121270

11

TECHNICAL REPORT RD-CR-82-26

SOFTWARE DEVELOPMENT FOR THE ANALYSIS OF IMPROVED HAWK WEAPONS SYSTEM

Gerald Johnson, Laura Jeziorski, and James Marr
School of Engineering
The University of Alabama in Huntsville
Huntsville, Alabama 35898

July 1982

Prepared for
System Simulation and Development Directorate
US Army Missile Command
Redstone Arsenal, AL 35898

U.S. ARMY MISSILE COMMAND
Redstone Arsenal, Alabama ~~35809~~ 35898

Cleared for Public Release; Distribution Unlimited.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RD-CR-82-26	2. GOVT ACCESSION NO. AD-A121 270	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Software Development for the Analysis of Improved HAWK Weapons System		5. TYPE OF REPORT & PERIOD COVERED Final Tech Rpt 4/81-6/82
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gerald Johnson, Laura Jeziorski, James Marr		8. CONTRACT OR GRANT NUMBER(s) DAAH01-81-D-A006
9. PERFORMING ORGANIZATION NAME AND ADDRESS School of Engineering The University of Alabama in Htvl Huntsville, AL 35898		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Missile Command Redstone Arsenal, AL 35898 DRSMI-RD		12. REPORT DATE July 1982
		13. NUMBER OF PAGES 46
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Cleared for Public Release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

FEASIL	relational
Data Base	simulation
IHAWK	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The FEASIL data base relational management system is discussed. The current implementation of the FEASIL system for the IHAWK missile project is discussed. The creation and operation of the IHAWK data bases is detailed. Conclusiions and recommendations for further development of FEASIL and IHAWK data base systems are presented.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

PREFACE

This technical report was prepared by the Research Staff of the Electrical Engineering Department, School of Engineering, The University of Alabama in Huntsville. The purpose of the report is to provide documentation of the technical work preformed and results obtained under delivery order 0005 of MICOM Contract No. DAAH01-81-D-A006; Dr. N. A. Kheir, Principal Investigator.

The project documented herein was performed by G. E. Johnson, Dr. J. D. Marr and L. D. Jeziorski. Dr. M. M. Hallum III, Chief, Systems Evaluation Branch, Army Missile Laboratory, U. S. Army Missile Command, was technical monitor, and Mr. J. R. Davis provided technical coordination.

The authors wish to acknowledge the valuable discussions and assistance provided throughout the task by Dr. M. M. Hallum, III, J. M. Jordan, and J. R. Davis, of the Systems Evaluation Branch, D. D. Tarkowski and M. A. Strickland of Georgia Tech, and H. Richard of Boeing.

The technical viewpoints, opinions, and conclusions expressed in this report are those of the authors and do not necessarily express or imply policies or positions of the U. S. Army Missile Command.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

The advent of advanced digital technology and simulation techniques has permitted numerous digital and hardware in the loop simulated test flights to be performed before actual missile test launches. This advance in technology has aided in reducing research and development costs of missile systems. At the same time it has provided enormous amounts of flight test data, generated during a relatively short period of time. However, the increased speed and reduced cost of research and development have yet to remove the necessity to execute data reduction and correlation processes. These processes are essential in order to allow modifications which eliminate unnecessary duplicate test runs. To provide for the reduction and correlation requirements, various data base system structures have been developed.

The IHAWK missile system evaluation project relies heavily on digital and hardware in the loop simulations to test various independent subsystems as well as the complete system as a unit. The data reduction techniques and processes utilized during previous testing periods have proven to be time consuming and laborious.

This report will document the effort to implement a data base structure which supports the IHAWK testing process. The data base structure is implemented on a PERKIN-ELMER OS 8/32 computer.

The first section of this report will familiarize the reader with data base systems in general, and the relational data base system in particular. Individuals already acquainted with the relational concept and the concepts of the "FEASIL" Data Base Management System should proceed to the next section. Section 3 provides a detailed explanation of how the FEASIL data management system was utilized to create several data bases for IHAWK test data. Current listings for the various programs used to transfer test data from the CDC 6600 to the PERKIN ELMER system and to structure the associated test condition information into a FEASIL computable format are contained in the Appendices. Section 4 presents conclusions about the efficiency and effectiveness of the current data base structure. Recommendations for improvements in the reduction process and FEASIL utilization are included also in Section 4.

2.0 THE FEASIL DATA BASE SYSTEM

2.1 INTRODUCTION

The FEASIL data base management system is an important tool for the organization and analysis of data generated in the HAWK Weapon System program. These data include system response, sensor outputs, and many other types of information. If the data is not organized in a conveniently recoverable form, then much of its value is lost. The FEASIL relational data base system is intended to serve as a solution to this problem.

In Section 2.2, data structures with particular emphasis on relational data structures is examined. A less-technical, intuitive approach is used to aid those not familiar with the area. Section 2.3 describes current aspects of the FEASIL system and directions for natural evolution. Suggested areas for the enhancement of FEASIL system are summarized in Section 2.4.

2.2 RELATIONAL DATA BASE SYSTEMS

The relational data structure is based on a "flat file" principle. The structure conceptually resembles a table, with a heading across the top and one record (or tuple) per line downward. Headings correspond to the item names within each tuple. Items listed on each line of the table correspond to items in the record.

The term "record" refers to a set of related items stored in a file as a logical record. A "tuple" is that same set of items with no consideration of how they are stored. The meaning of any item depends only on its position in the record and on the meaning defined for that position in the record. Logical record size is frequently related to hardware structure and physical record size of the computer or of some other device. For example, some systems store programs in 80-character physical records to resemble 80-column punched cards. Similarly, physical records of 256 characters (64 floating point numbers) are popular due to disk structures. A data base structure may use logical records of any size, but some benefit may be gained (in input/output speed) if logical record size is a multiple of physical record size.

Items in a record may be of any standard type, such as integer or text. A record is of predetermined size, set at creation, so a fixed amount of space must be allowed for each item. On the Perkin-Elmer system, a memory word can hold an integer number, a floating point number, or four characters. On a 16-bit minicomputer, integers usually require one word and floating point numbers require two; a word can hold two characters. Thus, size of the record depends on the number and types of data items stored therein.

Ignoring any hardware dependent restrictions, most data structures can be converted (from other forms) to the relational data structure at the cost of some increase in redundancy. For example, consider the data in Figure 2.1, showing two relational structures and one possible hierarchal structure for the same information. The first relational structure of Fig. 2.1(A) shows all the information in a single table with each row (or tuple) referring to one President. Several Presidents came from the same state or from the same party, so it is possible to reduce storage space by assigning each possibility a code number and using a look-up table. A multiple relations structure is depicted in Fig. 2.1(B). It requires less space overhead to build a second table than to write out "Virginia" or "Massachusetts" several times. This is still a relational system, but uses three relations to convey the same information.

In a relational system, actual storage order in the data base may be chosen to minimize search time. If searches are typically for "the first president who ...," then chronological order is appropriate. Actual order of records or tuples in the database file is irrelevant to the user; only the order of printing matters. It could be just as useful to some users to have printed the rows in alphabetical order on column 2 (name). It is also not necessary to print all the columns in the relation. If presidents are being listed by home state, only the state and name may be desired.

It is possible to prepare a relation with a seperate tuple for each term of each president. This would include about 50 lines including five lines for FDR. To save space we could then shift from a relational table structure to a tree structure as shown in Figure 2.1(C). One record type would give the number, name, party, state, and pointers to the records for terms served. A second record type would have the starting date for each term served. Note that in this structure, each "offspring" (year) has only one "parent" (name record). Other structures possible are illustrated in Figure 2.2, which is adapted from Principles of Data-Base Management by James Martin. A plex structure, for example, would be well adapted for relating presidents to parties. The tree structure can be more effective in showing precedence, such as in organizational charts. Advantages of the relational structure over hierarchal structures include ease of use, flexibility, ease of implementation, and clarity. However, hierarchal forms have been understood longer, and consequently most data base systems have been designed around the earlier structures.

For some applications, such as payroll and personnel records, the level of redundancy is even higher than in the presidential example, and a multiple relational structure is more effective. For example, changes are easier if you can alter the pay level for "GS-5, step 7" in one place, rather than search for all persons of that rank and change the item in their individual records. For this type application, the multirelation system is greatly superior to a single relation. For scientific applications, there is usually much less redundancy and the single relation is more appropriate. Each record is more likely to refer to a unique event rather than to one element out of a set of nearly identical cases.

Number	Name	Party	Started	State
1	Washington, G.	Federalist	1789	Virginia
2	Adams, J.	Federalist	1797	Massachusetts
3	Jefferson, T.	Republican*	1801	Virginia
4	Madison, J.	Republican*	1809	Virginia
5	Monroe, J.	Republican*	1817	Virginia
6	Adams, J. Q.	Republican*	1825	Massachusetts
7	Jackson, A.	Democrat*	1829	South Carolina
.....				
35	Kennedy, J.	Democrat	1961	Massachusetts

Figure 2.1(A) Relational Structure

Relation 1

Number	Name	Party Code	Started	State Code
1	Washington, G.	1	1789	1
2	Adams, J.	1	1797	2
3	Jefferson, T.	2	1801	1
4	Madison, J.	2	1809	1
5	Monroe, J.	2	1817	1
6	Adams, J. Q.	2	1825	2
7	Jackson, A.	3	1829	3
....				
35	Kennedy, J.	2	1961	2

Relation 2

Party Code	Party
1	Federalist
2	Republican until ..., now called Democrat
3	Democrat until ...

Relation 3

State Code	State
1	Virginia
2	Massachusetts
3	South Carolina

Figure 2.1(B) Multiple Relations

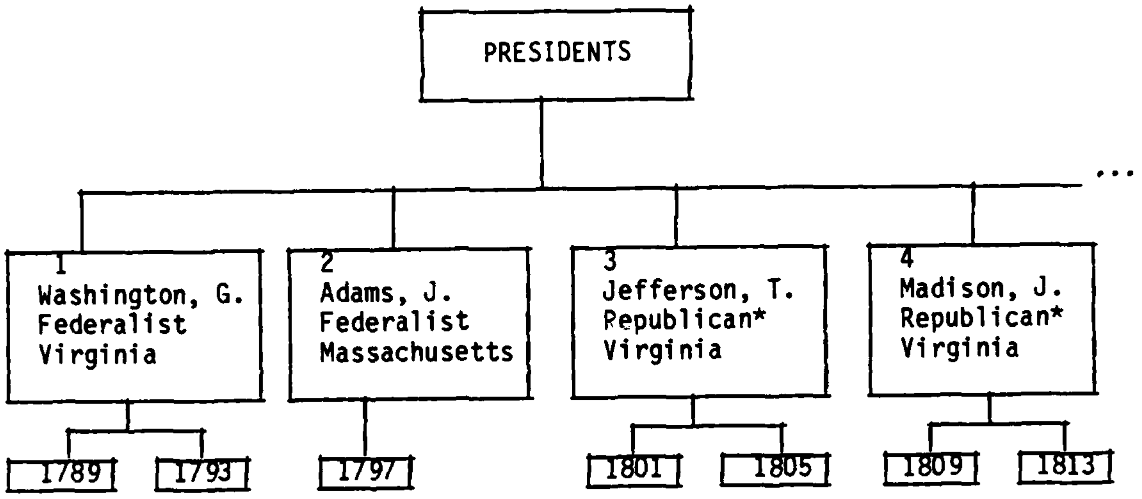
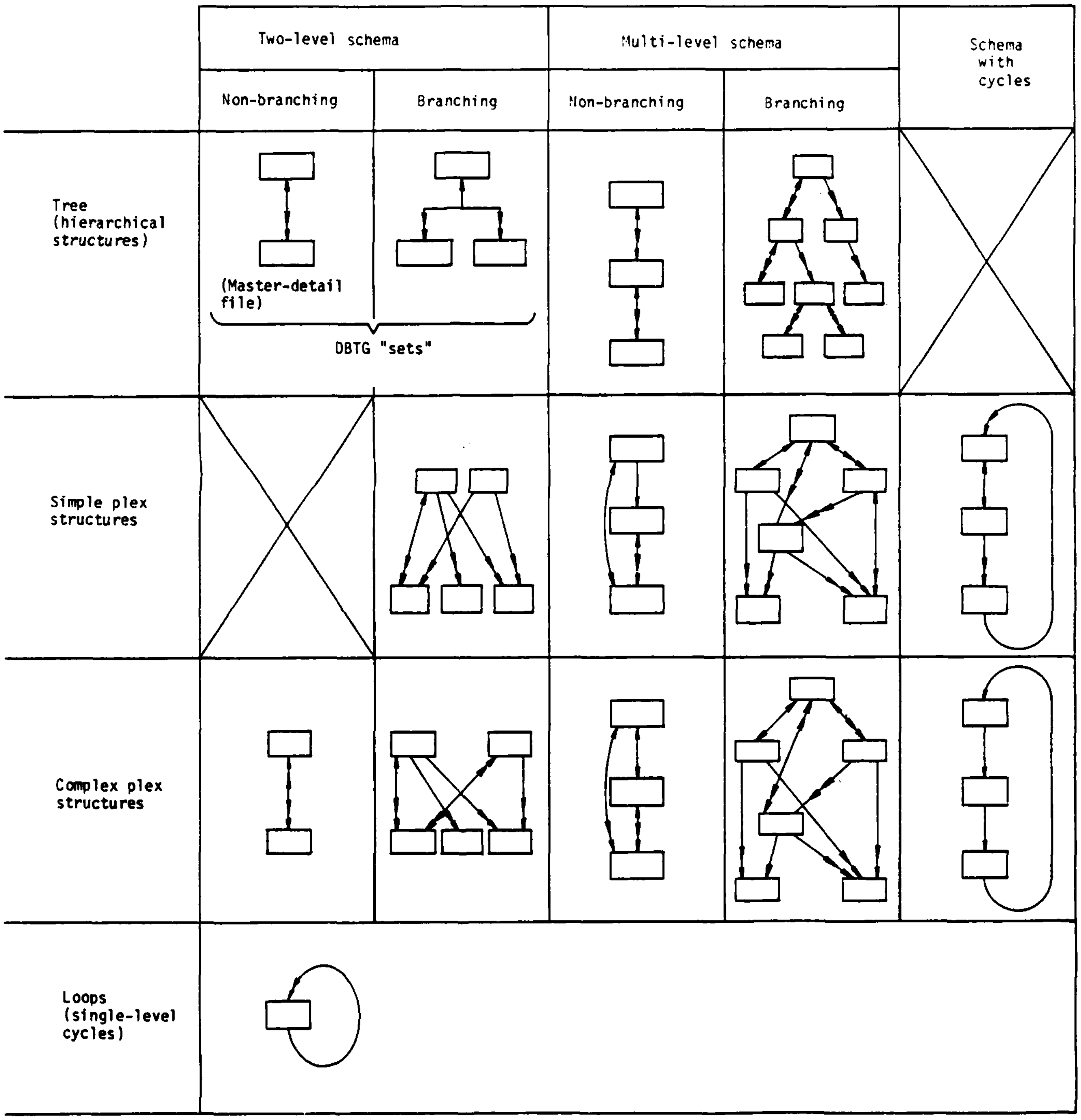


Figure 2.1(C) Hierarchal Structure



Categories of schema. Data-base management systems and languages differ in which of these structures they can handle. Some can handle hierarchical structures but not plex structures. Some can handle simple plex structures but not complex plex structures. The number of levels that can be handled differs from one system to another. Few can handle loops.

Figure 2.2 Other Structures

From the Principles of Data Base Management by James Martin

2.3 THE CURRENT STATE OF FEASIL

A complete description of FEASIL structure and operation is beyond the scope of this report. For more detail, the reader should consult Dr. Hallum's original report [3]. This section will consider only details of the current capabilities and the file structure of FEASIL Revision 6. FEASIL 6 is compiled under Fortran 6.

FEASIL 6 has a total of ten major functions, nine of which are public; the tenth, print a file, is for use only by system programmers. The public options are as follows:

1) Create a new relation (with the full set of files)
2) Delete a relation
3) Edit a relation
4) Modify column specifications
5) Merge two relations (into a single relation)
6) Reorganize a relation
7) Retrieve and manipulate data
8) Back up a relation (save a backup copy)
9) Status a database

Function 10 is not public.

The FEASIL system uses three files for a relation: the tuple descriptor file (TDF), the tuple file (TF), and the alpha data file (ADF). The TDF contains the description of the relation using one array to store the type and title of each column in the relation and another to store size information. The TF contains the tuples, stored one per logical record. The ADF stores all text of more than a single character, at one item per record. The TDF has pointers to the ADF for records in which column names are stored. The TF similarly has pointers to ADF records for any item that is in a multicharacter type column.

Four data types (strategies) are supported: single character, integer, floating point, and arbitrary-length string. Arrays and complex or double precision variables are not supported.

As stated earlier, the TDF consists of two arrays. One holds information on the current status of the relation, such as the sizes of the three files, the number of tuples and columns, the next record number to use in each file (because the file is not full), the number of deleted columns or tuples not yet removed by a repack operation, etc. The other array is 5 by N for the N columns of the relation. For each column, this array stores the strategy of the column, the length and ADF record number of the column's name, and the column position within the record. The TDF information is always kept in core.

The TF holds one tuple per record, and only the record currently being used is kept in core. Record length depends on the number and strategies of items in the tuple. The arbitrary-length string strategy requires two words for the length and pointer to the ADF block; the other strategies require one word for the value. All data is handled as Fortran integer, with appropriate type conversions.

The ADF uses 256 byte records for storage of string data. Each record consists of one word (four bytes) to indicate string length and the remaining 63 words are used for characters; maximum allowable string length is therefore 252 characters. If a two-character string is stored, it still requires an entire block. Several alternate strategies are possible, and are indicated as directions for later improvement [3]. Only the current ADF record resides in core. If a record in the ADF has been deleted, it is indicated by storing the four-character string "FFFF" in the record.

Structurally, FEASIL consists of a root program and three overlays. About 60 common areas are shared among the main program and about 80 subroutines and functions.

2.4 EVOLUTION OF FEASIL

There are several directions for natural evolution of the FEASIL system. These may be divided into language-related changes, capability enhancements, and friendliness.

FEASIL currently runs under Fortran rev 6, a Perkin-Elmer enhancement of Fortran IV. There are five assembly language subprograms. A FLEX preprocessor is used to convert from a structured variant of Fortran into a standard Fortran code. Recently, the new standard Fortran 77 has been developed, enhanced by more powerful programming structures such as the IF-THEN-ELSE and by a character data type. This language is available as Fortran 7 on the Perkin-Elmer system.

For portability to other systems, it would be useful to convert FEASIL into a standard Fortran 77. This would require substantial effort in the replacement of specialized FLEX structures, but could be implemented gradually on a module-by-module basis. One of the most urgent portability problems is replacement of the assembly language routines with standard Fortran routines. Fortunately, it appears that the current assembly language routines may be functionally replaced with short Fortran 77 routines. Finally, since Fortran 77 supports logical and relational operations on the character data type, data handling methods for character data could be updated. The main technical reason for delaying conversion would be the lack of Fortran 77 compilers on some computers.

Evolutionary enhancement of features of FEASIL affects data types and data manipulation capabilities. Handling of character string could be improved and input-output options could be expanded.

As mentioned earlier, the current ADF structure lacks flexibility. Strings may not exceed 252 characters, but short strings waste most of the space in a standard block. Possible solutions include packing more than one item into a block, or shortening the block and allowing multiple-block items.

As an independent action in string handling changes, a short string strategy could be allowed which would not use the ADF and would store four or eight characters. This would satisfy some of the demand for string variables, and take no more TF space than the present unrestricted-length string. If FEASIL were used for a memo filing system, for example, the ability to store DRSMI or -RD in a single item without ADF use could be valuable, both for speed of searching and for conservation of disk space. There are similar scientific uses for a short string strategy.

There are currently two ways to enter data into FEASIL. The first method is to type it into the program, one item at a time. The second approach uses a new data entry option within the edit command [4] The new option allows input of free format data from card images, separating the items with a user-specified delimiter. This still leaves other input options to be implemented, such as ASCII formatted input or binary input.

Output is also limited. Tabular output is available, but more flexibility could be added. Graphical output to the screen and to the line printer could be highly useful, even if only of low resolution. Plots of time functions or scatter plots would be a reasonable starting point. The only functions now available are sum, mean, and standard deviation. Other statistical analysis capabilities and other data manipulation techniques would also be useful.

User-friendliness is more difficult to define. Among other areas, it deals with the phrasing of prompts so that a novice user can operate the system without excessive difficulty. It also deals with convenience of use, tolerance to improper inputs, and general interfacability. As an example, the message "this sort will take about 200 minutes; do you really want it done?" could be replaced by a suggestion on how to alter the relation to achieve faster sorts (after a "no" reply to the question above). The system should be simple enough for an executive to operate, but powerful enough to satisfy full-time users in search of greater capability. Section 4 presents our list of recommendations to enhance the FEASIL with a suggested time table.

3.0 IHAWK SUPPORT SOFTWARE

3.1 INTRODUCTION

The ability to reduce, coordinate, and analyze the vast amount of flight test data generated by an IHAWK simulation test run is critical to the effective and timely development of the missile system. With the goal of providing this ability, a series of relational data base structures were implemented on a PERKIN ELMER OS 8/32 system. The creation and operation of each data base is discussed in the following sections along with a general overview of the internal structure. The transfer of data from the CDC 6600 system to the PERKIN ELMER OS 8/32 system is detailed with listings provided in the Appendices.

3.2 IHAWK DATA BASES

There are currently four IHAWK Data Bases being utilized to contain the existing simulation data. These four data bases are summarized as follows:

Data Base A: This data base contains the reduced data for flight and end game conditions of an actual missile flight. These conditions were taken from the IHAWK flight register and are used to indicate test conditions for the hardware-in-the-loop simulation.

Data Base B: This data base contains the reduced data on simulation conditions and test results taken from the IHAWK tests conducted at the Boeing Company's terminal guidance laboratory located at Kent Space Center. The test period was from July, 1974 through December, 1975.

Data Base C: This data base contains the flight test conditions for a simulation presented in a coded format.

Data Base D: This data base is an experimental database containing an integration of simulation conditions and simulation test results produced by the Radio Frequency Simulation System (RFSS) facility.

3.2.1 DATA BASE A

Data Base A contains reduced data for flight conditions previously mentioned. This data base consists of the following relations.

Relation Name	Brief Description
IHFTTEST	Preflight flight and end game conditions
IHFTTYPE	Type Telemetry
IHPURPOSE	Purpose Designation
IHFTLOC	Firing Location Identifier
IHTGTTYP	Target Type Identifier
IHTGTAUG	Target Augmentation Identifier
IHFIREUT	Firing Unit Identifier
IHFTTERM	Termination Identifier
IHFUZE	Fuze Performance Identifier

Table 3.1 Data Base A Relations

The last eight relations indicated in Table 3.1 correspond to columns 2, 3, 4, 8, 9, 10, 18, and 19 respectively. These columns are a subset of relation IHFTTEST. Each relation contains the interpretations of the test codes used in each respective column. Table 3.2 contains a listing of the relationship between each relation and its corresponding column.

Relation	Column
IHFTTYPE	2
IHPURPOSE	3
IHFTLOC	4
IHTGTTYP	8
IHTGTAUG	9
IHFIREUT	10
IHFTTERM	18
IHFUZE	19

Table 3.2 Relation/Column Relationships

Current status of the relation IHFTTEST can be verified by utilizing the status command of the FEASIL data management system. An example of this method is depicted in Figure 3.1.

```
*FEASIL
- *°*°* You Are In DMS(VER F6.05).  Do You Want To:
- 1)Create A New Relation
- 2)Delete A Relation
- 3)Edit A Relation
- 4)Modify Column Specifications
- 5)Merge Two Relations
- 6)Reorganize A Relation
- 7)Retrieve & Manipulate Data
- 8)Backup A Relation
- 9)Status A Database
-  Enter Selection By Number >
->9
-
- Name Of Relation To Be Statused?>
->IHFTTEST
- Is The System Volume Being Used?
- <Yes Or No>
->No
- What Is Name Of Volumn You Are Using?
->MT5A
-
- Is Printer Copy Desired?>
-       <Yes or No>
->No
-
-             *°*°* Status Data For Relation IHFTTEST*°*°*
- Relation Consist Of     56998 Characters
- Relation Has:  28 Columns And   245 Rows.
- Dead To Active Space % =    0.00
- Do You Wish To Continue In Database System?
- <Yes or No>
->

- Symbol indicates the computer system response
-> Symbol indicates user response
```

Figure 3.1 Data Base A Status

Detailed internal structure for IHFTTEST is depicted in Table 3.3 which indicates each column name and associated strategy. This strategy indicates the type of variables allowed as inputs to the relation.

Col. No.	Col. Name	Strategy
1	Missile Serial Number	Integer
2	Type Telemetry	Integer
3	Purpose Designation	Integer
4	Firing Location Identifier	Integer
5	File Number	Integer
6	File Character	Single Character
7	Date Fired	Integer
8	Target Type Identifier	Integer
9	Target Augmentation Identifier	Integer
10	Firing Unit Identifier	Integer
11	Target Speed at Intercept m/sec	Floating Point
12	Target Slant Range at Launch-KM	Floating Point
13	Target Slant Range at Intercept-Sec	Floating Point
14	Target ALT at Intercept-AGL-KM	Floating Point
15	Missile Time-of-Flight to Intercept-Sec	Floating Point
16	Missile Time-of-Flight to FLT Termination -Sec	Floating Point
17	Intral Speedgate Lock Time-Sec	Floating Point
18	Termination Identifier	Integer
19	Fuze Performance Identifier	Integer
20	Doppler Miss Distance-M	Floating Point
21	IGOR Miss Distance-M	Floating Point
22	Test OBJ Results: (A-Achieved: N-No Test; F-Failure)	Single Character
23	Failure Category	Character String
24	Telemetry Performance: (G-Good; B-Bad; M-Marginal)	Single Character
25	Failure Reason	Character String
26	Failure Area	Character String
27	Flight Objective	Character String
28	Comments	Character String

Table 3.3 Relation IHFTTEST

3.2.2 DATA BASE B

Data Base B contains the Boeing simulation results. These results are contained in two main relations--MICOMCON which contains a run condition matrix and MICOMDB which contains the result matrix. These main relations are subdivided into twenty subrelations. These subrelations are indicated in Table 3.4.

Relation Name	Brief Description
R1	General description of design test conditions
R2	Design intercept condition
R3	Launch condition
R4	Target spatial configuration
R5	Target velocity
R6	Target fading and jockeying characteristics
R7	Missile condition
R8	Special equipment condition
R9	Hardware/software configuration
R93	Software configuration
R97	Hardware configuration
R10	Maneuver description
R11	Maneuver parameters
R12	Target and jammer configuration
R13	Jammer power levels
R14	Primary test jammer parameters
R15	Swept jammer parameters
R16	Deceptive jammer parameters
R17	Missile HOJE state
R18	Necessary comments
*MICOMCON	Run condition matrix
*MICOMDB	Result matrix

*Denotes main relations

Table 3.4 IH Data Base Configuration

The current status of the MICOMCON and MICOMDB relations are shown in Figure 3.2.

```
-
-        *°*°* Status Data For Relation MICOMDB *°*°*
- Relation Consist Of    271882 Characters
- Relation Has:  6 Columns And 9704 Rows.
- Dead To Active Space % =   8.83
- Do You Wish To Continue In Database System?
- <Yes or No>
->
```

Figure 3.2 MICOMDB Status

The internal structures of relations MICOMCON and MICOMDB are given in Table 3.5 and 3.6.

Col. No.	Col. Name	Strategy
1	Date	Integer
2	Number	Integer
3	1	Character String
4	2	Integer
5	3	Integer
6	4	Integer
7	5	Integer
8	6	Integer
9	7	Integer
10	8	Integer
11	9	Integer
12	10	Integer
13	11	Integer
14	12	Integer
15	13	Integer
16	14	Integer
17	15	Integer
18	16	Integer
19	17	Integer
20	Comments	Character String

Table 3.5 Relation MICOMCON

Col. No.	Col. Name	Col. Strategies
1	RR-1	Integer
2	RR-2	Integer
3	RR-3	Integer
4	RR-4	Integer
5	RR-5	Floating Point
6	Number	Integer

Table 3.6 Relation MICOMDB

3.2.3 DATA BASE C

Data Base C contains the flight test conditions in coded format. The main relation for this data base is called MARK. This relation contains the SIO test matrix developed by Georgia Tech. Current status of this data base is outlined in Figure 3.3.

```
-
-        *°*°* Status Data For Relation Mark *°*°*
- Relation Consist Of    29721 Characters
- Relation Has:  21 Columns And   315 Rows.
-
- Dead To Active Space % =   0.00
-
- Do You Wish To Continue In Database System?
- <Yes or No>
->
```

Figure 3.3 Data Base C Status

The current internal structure of this relation is illustrated in Table 3.7 which outlines the appropriate column names and strategies.

Col. No.	Col. Name	Strategy
1	CC	Integer
2	D	Integer
3	FRN	Integer
4	TRN	Integer
5	M	Integer
6	R	Integer
7	X	Integer
8	P	Integer
9	S	Integer
10	TC	Integer
11	SL	Integer
12	BP	Integer
13	DC	Integer
14	Notes	Character String
15	GT	Integer
16	T/T	Integer
17	EM/B	Integer
18	HS	Integer
19	VS	Integer
20	MS	Integer
21	ST	Integer

Table 3.7 Relation MARK

3.2.4 DATA BASE D

Data Base D is an experimental relation called MISSDIST. This relation was created from a storage file using the ADD command in the FEASIL edit mode. Figure 3.4 indicates the current status of this relation.

```
-
-          *°*°* Status Data For Relation MISSDIST *°*°*
-
- Relation Consist Of     396700 Characters
-
- Relation Has:   9 Columns and 2734 Rows.
-
- Dead To Active Space % =     0.00
-
- Do You Wish To Continue In Database System?
- <Yes or No>
->
```

Figure 3.4 Data Base D Status

The internal structure of this relation is depicted in Table 3.8.

Col.	Col. Name	Strategy
1	Date	Integer
2	Run No.	Integer
3	Cond. Code	Character String
4	XTVEL	Real
5	YTVEL	Real
6	ZTVEL	Real
7	TSTOP	Real
8	PMISS	Real
9	Comment	Character String

Table 3.8 Relation MISSDIST

3.3 CREATION OF DATA BASE D

The following describes the procedure developed to transfer data generated in the RFSS simulation into a data base so that data can be easily retrieved and manipulated. Data generated in the RFSS is stored in a permanent file on the CDC 6600 computer. Since the data base management system is stored on the PERKIN ELMER (P. E.) Interdata 8/32, the data must be transferred between systems. The FORTRAN program listed in Appendix A is used to enable the CDC to read the data file and write the data in BCD to 7-track tapes at 800 b.p.i. and 132 byte records in a specified format (See Fig. 3.5). The tapes are then placed on the P. E. Interdata 8/32 7-track tape drive at 800 b.p.i. and the program listed in Appendix B is used to copy the data from the tapes to a data file stored on disk using the P. E. copy mode. The file name is specified by the user, (A private disk must be used to store the data file because the data files are generally too large to store on the system disk core). This program also loads the task of a FORTRAN program called HAWK2.FTN (listed in Appendix C) which sorts the data and combines it with another file which contains codes for the scenerio conditions (condition code file) of the simulation. The condition code file must be created before running this program. It must be created in the format which the program will later read. This format is discussed later. FEASIL can then be used to read the file containing data that is in 80 column card format and put it into an empty relation. This relation must be created prior to running the programs.

The procedure utilized to transfer data generated during a simulation run in the RFSS into a data base is illustrated in the following six-step process.

Step 1: A test condition code file is created in the PERKIN ELMER editor.

Step 2: The simulation data is transferred from the permanent file located on the CDC 6600 to 7 track tapes formatted for PERKIN ELMER 8/32 usage.

Step 3: Simulation data on the 7 track tapes is transferred to a permanent file on the PERKIN ELMER OS 8/32.

Step 4: The test condition code file and the simulation data file are combined using a Fortran driving routine.

Step 5: The desired FEASIL relation is created.

Step 6: The file produced from Step 4 is entered into the relation created in Step 5 using the FEASIL ADD command.

Next, each of these steps will be examined in detail.

```
 138          26           6          81           0           0           2           0           0           1
   0           1           1           1           0           0           0           1           8           0
   0           0           0           0           1
  24689.00000             -249.99219             -2999.99219              -374.99707                 0.00000
      0.00000                0.00000                 0.00000                 0.00000                 0.00000
      0.00000                 .04761                 0.00000                  .69995                  .39063
       .69995                 .39063                46.09863               107.85278                 7.81250
     46.09863              107.85278                 7.81250                 0.00000                 0.00000
      0.00000                0.00000                 8.41504                 0.00000                -3.57715
     23.02998                0.00000                 0.00000                 0.00000                 0.00000
      0.00000                0.00000                90.00028                15.34330                 -.37432
      0.00000               29.99973                26.31999             14819.43750              -249.99219
  -2999.99219             -374.99707                 0.00000                 0.00000                 0.00000
  14249.46875              -22.43750             -2856.76563               490.01074               -38.84277
     11.74316                1.08035                -6.56446               -10.14027                40.31239
     28.89796                2.97676                 3.29906                90.00028                 2.30633
      -.00804               26.98671                 4.37500              -236.67188              -186.02344
    301.06250                4.37500               263.32031              -186.02344               322.40625
      0.00000                0.00000                 0.00000                 0.00000                 0.00000
      0.00000                0.00000                 0.00000                 0.00000                 0.00000
      0.00000                0.00000                 0.00000                 0.00000                 0.00000
      0.00000                0.00000                 0.00000                 0.00000                 0.00000
      0.00000                0.00000                 0.00000                 0.00000                 0.00000
```

Figure 3.5 CDC to Tape Format

Step 1: Create the condition code file

The condition code is a 10-character code with each character being coded to represent specific scenario conditions. Georgia Tech used the following code to construct the condition code to be entered into the database:

The first character is downrange intercept.

The second character is centroid crossrange intercept.

The third character is intercept altitude.

The fourth character describes target separation (M) and orientation:

The fifth character describes target maneuver parameters.

The sixth character describes Jammer and other states.

The seventh and eighth characters describe Jammer One.

The ninth and tenth characters describe Jammer Two.

Table 3.7 lists the runs which were coded into their condition codes:

Date	Number of Runs	Number of Runs Analyzed
25 June	56	15-56
26 June	405	All
29 June	161	All
30 June	162	All
1 July	--	1-391
2 July	--	1-45
30 July	185	166-185
31 July	401	1-100, 266-291
3 August	--	1-75
4 August	80	All
5 August	356	All
6 August	1-266	1-205
	357-261	357-361
7 August	650	All

Table 3.7 Condition Coded Runs

The Condition Code File is created using the P.E. 8/32 editor. In addition to the 10 character condition code, the file also contains the run number, comments and data corresponding to the condition code. The file is formatted as follows:

Starting in column (1):	Run Number
Starting in column (6):	Condition Code
Starting in column (17):	Comments
Starting in column (60):	Year/Month/Day

Step 2: Transfer from CDC to 7-Track Tapes

It is assumed that the reader is familiar with running a batch job on the CDC 6600. The program (Appendix A) reads from a permanent file on the 6600 and writes to a 7-track tape in the following format:

2 lines of 10 integers 10 digits long with each integer separated by one space; 1 line of 5 integers 10 digits long with each integer separated by one space; 20 lines of 5 real numbers 20 digits long with each real number separated by two spaces.

The data is written in BCD at 800 b.p.i. in 132 byte records. See Figure 3.5 for an example of format. The 7-track tapes are then transferred to the P.E. 8/32.

Step 3: Transfer from 7-Track to PERKIN ELMER Interdata

This step is accomplished by a CSS procedure. This procedure uses the OS/32 to copy the data from the 7-track tape to a file name specified by the user. The procedure is stored in a file called RRFSS.CSS and is run as follows:

```
Enter >(Name of CSS procedure) (filename)
                    or
Enter >RRFSS (filename)
```

See Appendix B for listing of RRFSS.CSS. For explanation of Command Substitution System see PERKIN ELMER OS/32 Operator Reference Manual, 529-574 R03.

Step 4: Combine Files

The CSS procedure RRFSS also runs a fortran program which sorts the simulation data and combines it with the condition code file. This is accomplished by testing for good runs and writing the good runs to a temporary file. The temporary file is then matched by run number and date to the condition code file. Specified variables are then written to the final file in a format that FEASIL can read. The program is filed under HAWK2.FTN (Appendix C). This is an interactive program and asks the user for the following:

1) Filename of condition code file
2) Filename of combined file
3) Filename of simulation data file

Step 5: Create a Relation

A data base should be designed in such a way as to meet the needs of those who will be using the data base. To prevent confusion, mistakes, and wasted time, column names and strategies should be decided before going to FEASIL.

The next step is to go to a terminal connected to the P.E. 8/32 which is located on the 3rd floor of building 5400. This is where FEASIL software is presently stored. Refer to page 29 on the procedure to signon to the terminal. Refer to page 30 on loading FEASIL software.

Col. No.	Col. Name	Col. Strategy
1	A	Integer
2	B	Integer
3	C	Floating Point
4	D	Character String
5	E	Character String
6	F	Single Character

Table 3.8 Relation Test

The following example illustrates the process of creating a relation with the configuration outlined in Table 3.10.

```
FEASIL
  *°*°* You Are In DMS(VER F6.06).  Do you want to:
  1) Create a New Relation
  2) Delete a Relation
  3) Edit a Relation
  4) Modify Column Specifications
  5) Merge Two Relations
  6) Reorganize a Relation
  7) Retrieve & Manipulate Data
  8) Backup a Relation
  9) Status a Database
  Enter Selection By Number >
 >1
  Name Relation
 >Test
  Confirm Creation of Relation Test
  <Yes or No>
 >Yes
  Is the System Volume Being Used?
  <Yes or No>
 >No
  What is Name of Volume Being Used?
 >MT6A
  Confirm Read Key 0 and Write Key 0 on Storage Volume MT6A
  <Yes or No>
 >Yes
  Number of Columns (Order) of Relation?>
 >6
  Supply Column Names & Strategies as Indicated.
   Column Strategy Choices Are:
      1 - Integer Number
      2 - Floating Point Number
      3 - Single Character
      4 - Character String
  Column      1
      Column Name?>
  >A
      Strategy?>
  >1
  Column      2
      Column Name?>
  >B
      Strategy?>
  >1
```

```
Column     3
    Column Name?>
>C
    Strategy?>
>2
Column     4
    Column Name?>
>D
    Strategy?>
>4
Column     5
    Column Name?>
>E
    Strategy?>
>4
Column     6
    Column Name?>
>F
    Strategy?>
>3
Column     1
    Column Name> A
    Strategy> 1-Integer
Column     2
    Column Name> B
    Strategy> 1-Integer
Column     3
    Column Name> C
    Strategy> 2-Floating Point
Column     4
    Column Name> D
    Strategy> 4-Character String
Column     5
    Column Name> E
    Strategy> 4-Character String
Column     6
    Column Name> F
    Strategy> 3-Single Character
Any Corrections? <Yes or No>
>No
```

For additional reading see Technical Report RD-81-16 "Addition of Data Features for FEASIL."

Step 6: Send Combined File into FEASIL

The final file is now in a format that FEASIL can read (80-column card format). This means that each line in the file represents a card image. Also each element must be separated by a delimiter specified by the user.

Example: FEASIL will read each element between the delimiters as elements of a column where element 1 corresponds to col. 1 etc.

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5	Col. 6
Integer	Integer	FL.P.	String	String	Single Charac.

The following card images may be used to input data into this relation.

```
Card 1      1, 33, 5.3, Command Destruct, Autopilot, G
Card 2      5
Card 3      15, 12.4, Self Destruct, S+A, L, 7, 18
Card 4      8.1 None, Empty, Q
```

These four cards will result in 3 tuples added to the example relation. The delimiter in this example is the comma. It is imperative that the user remembers that there is an automatic delimiter in column 80 except for string characters.

The ADD command in FEASIL edit mode is used to input data from a file into a relation previously created. The following is an example taken from the above relation and data.

Example of ADD command:

```
FEASIL
*°*°* You Are In DMS(VER F6.06).  Do You Want To:
1) Create a New Relation
2) Delete a Relation
3) Edit a Relation
4) Modify Column Specifications
5) Merge Two Relations
6) Reorganize a Relation
7) Retrieve & Manipulate Data
8) Backup a Relation
9) Status a Database
 Enter Selection By Number >
>3
Name of Relation to be Edited?>
>Test
Is The System Volume Being Used?
<Yes or No>
>No
What is Name of Volume You Are Using?
>MT6A
```

```
    Relation Test Has:       0 Records     6 Columns.
    Current Record:      0
    Current Column:      1

    Display Column Names?<Yes or No>
    >Yes

    Display How Many Columns?>
    >6
    Display Column Number  1?>
    >1
    Display Column Number  2?>
    >2
    Display Column Number  3?>
    >3
    Display Column Number  4?>
    >4
    Display Column Number  5?>
    >5
    Display Column Number  6?>
    >6
    Ready to Edit>
    >
    >A
    Enter Number of Rows to be Added?>
    >4
    Enter Device or File Where Data is to Come From?
    >TESTFL.DAT
    Enter Delimiter Between Data Elements?>
    >,
```

This command can be used to input data into a relation already containing data or into an empty relation. For more explanation of this command see [4].

3.4 MANIPULATION OF A DATA BASE

This section will discuss how to use FEASIL to retrieve information from, manipulate, sort and edit data bases created by FEASIL. The IHAWK data bases relations MICOMCON, MICOMDB and IHFTTEST are presently located on the storage disk UAH1 under user account 61. Relations MARK and MISSDIST are stored under account 29. Therefore this section will also discuss the procedure for marking a disk on and off, signing on to the computer system, loading the FEASIL software package, and using certain FEASIL commands. Only those commands which are thought to be most used will be explained. For more information on the capabilities of FEASIL see [3]. Examples of each command and how to use them will also be given. All procedures discussed are relative to the Interdata P. E. OS 8/32 model presently being utilized.

3.4.1 MARK COMMAND

It is assumed that the user is familiar with the proper procedure of physically mounting a storage disk. The MARK command is used to enable usage of the disk volume. The following indicates the format for the MARK command.

```
               OFF
MARK dev:,             PROTECTED
                ON     SYSTEM
                       RESTRICTED=actno

                          bsize      exp

          CDIRECTORY =     80        100
```

Parameters:

dev:	is the device mnemonic.
ON	marks a device on-line.
OFF	marks a device off-line.
PROTECTED	marks a device write-protected.
RESTRICTED =	actno is the account number of the owner of the restricted disc. If a disc is marked on restricted, only the owner has access to it unless otherwise specified via the RVOLUME command under MTM.

SYSTEM	inhibits write access to a task running under MTM.
CDIRECTORY	creates a secondary directory.
bsize	is the number of files in each block of the secondary directory. bsize ranges from 20 through 5000; the default is 80.
/exp	is the expansion size for the secondary directory file on disc. This is the number of files that can be allocated before the secondary directory file overflows.

The MARK command must be entered from the system terminal. An example of the proper format for loading a disk onto disk drive No. 4 would be:

```
MA DSC4:, ON
     or   MA DSC4:, ON,,CD
```

The CD stands for Core Directory. This provides for faster access to the file directory of a disc volume. For further explanation see [5].

3.4.2 SIGNON COMMAND

SIGNON User-ID, Account Number, Password

The SIGNON command connects the user to the P. E. OS 8/32 capabilities through the multi-terminal monitor (MTM) software. The command must be in all capital letters.

```
Example:  S   TIM,61,
                or
             SIGNON  TIM,61,
```

This command can be typed in any terminal connected to the P. E. other than the system terminal.

3.4.3 LOADING FEASIL MANIPULATION MODE

The FEASIL software package is loaded and executed through a command substitution system (CSS) routine FEASIL.CSS. In order to execute the CSS routine the user enters the name FEASIL. The following example indicates the P. E. system and user interaction for manipulation mode.

```
*FEASIL
- *°*°* You are in DMS(VER F6.05).  Do you want to:
- 1)Create A New Relation
- 2)Delete A Relation
- 3)Edit A Relation
- 4)Modify Column Specifications
- 5)Merge Two Relations
- 6)Reorganize A Relation
- 7)Retrieve & Manipulate Data
- 8)Backup A Relation
- 9)Status A Database
-  Enter Selection By Number >
->7
-
- Name Of Relation To Be Manipulated?>
->SDLWS
- Is The System Volume Being Used?
- <Yes or No>
->No
- What Is Name Of Volume You Are Using?
->MT5A
- Ready For Manipulation>
->
```

After the user has entered the FEASIL manipulation mode the list of commands in Table 3.11 is available to retrieve and/or manipulate the data stored in a relation.

<Command>	<Function>
Q(Quit)	Terminates relation manipulation
C(Columns)	List names of each column by number in relation
R(Reproduce)	Copies relation under manipulation in to a relation specified by user.
P(Print)	Prints relations' data according to users' specifications
S(Sort)	Sorts a relation for printing purposes. Basic relation altered to last sort status
I(Re-initialize)	Initializes or re-initializes all records to active status
A(And)	Keeps active those active records that meet the selection criteria
O(Or)	Makes active all records that meet the selection criteria
M(Move)	Moves active records to relation designated by user
F(Function)	Takes user into area where operations such as sum average, and standard deviation may be performed on data.

Table 3.11 FEASIL Manipulation Commands

The following sections give further explanations and examples of the following commands:

1) And/Or/Re-initialize/Move
2) Function
3) Print
4) Quit

3.4.4 RETRIEVING INFORMATION (AND/OR/RE-INITIALIZE/MOVE)

All records are "active" if the AND or OR commands have not been used. If the AND or OR commands have been used, any following operations will be performed only on those records selected by the AND or OR command.

A(And)	This command keeps active those active rows which meet the selection criteria. The selection criteria available to the user are <,>, or =.
O(Or)	Makes active all rows that meet the selection criteria. The selection criteria for this is the same as the AND command. Rows that have been made inactive by the AND command can be made active again by the OR command.
I(Initialize)	Initializes or re-initializes all rows to active state
M(Move)	Moves active records to a relation designated by user

Usually after the ANDing and ORing process, the MOVE command is used to more easily examine and to keep track of the desired rows or records. Thus, the user may sort out pieces of the main relation.

Example: This example is an actual request that was made to retrieve information from the relation MICOMCON and MICOMDB. Relation MICOMCON consists of 20 columns and X number of rows. Certain columns are coded with integer values. The relation is to be sorted as follows.

1) Col. 10 must contain a 1 or 2
2) Col. 14 must contain a 6
3) Col. 16 must contain a 10, 11, or 12 and
4) Col. 19 must contain a 1

It is advised before getting started that a plan be made up as to the method and order of operations to be performed such as the following:

Requirement 1

1) On column 16 in MICOMCON perform the And operation for values greater than 9.

```
        A > 9
```

All tuples with a value > 9 in column 16 are kept active.

2) Move those active records to a new relation called MOVEONE.

```
              M    MOVEONE
```

3) On column 16 in relation MOVEONE perform the AND operation for values less than 13.

```
            A < 13
```

All rows with a value < 13 in column 16 are kept active.

4) Move those active records to a new relation called MOVETWO.

```
            M   MOVETWO
```

Now column 16 in MOVETWO has only the values (codes) 10, 11, and 12 (values > 9 and < 13) contained in it. The reason for moving the active records is to reduce the size of the relations, therefore decreasing the time it takes for FEASIL to complete one operation. This same result could be obtained by

```
    A(And) = 10         O(Or) = 11          O(Or) = 12.
```

<u>Requirement 2</u>

5) On column 10 in relation MOVETWO perform AND operation for values less than 3.

```
          A < 3
```

6) Move those active records to a relation called MOVETHREE.

This move and the following moves are necessary because each operation is performed on a different column.

```
               M    MOVETHREE
```

<u>Requirement 3</u>

7) On column 14 in relation MOVETHREE perform AND operation for values equal to six.

```
          A = 6
```

8) Move those active records to a relation called MOVEFOUR.

```
          M    MOVEFOUR
```

<u>Requirement 4</u>

9) On column 19 in relation MOVEFOUR perform AND for values equal to one.

```
          A = 1
```

10) Move those active records to a relation called MOVEFIVE.

```
          M   MOVEFIVE
```

The selection process is now complete.

Example of AND command:

```
    -  Ready for Manipulation>
    ->
    ->A
    -  Column on Which to Select?>
    ->1
    -  Selection Criteria (Options <,> OR =)
    -      (Which One)?>
    -><
    -  Value?>
    -  >6
    -  ANDing on Column  1 Complete>
    -
    -  Selection Process Complete-Active Records =   5
    ->
```

Example of OR Command:

```
    0
    - Column on Which to Select?>
    ->1
    - Selection Criteria (Options <,> OR =)
    -       (Which One)?>
    ->>
    - Value?>
    ->2
    - ORing on Column 1 Complete>
    - Selection Process Complete-Active Records = 16
    ->
```

Example of MOVE Command:

```
    ->M
    - Relation to Move Records to?>
    ->TSDLWS
    - Is the System Volume Being Used?
    - <Yes or No>
    ->No
    - What is Name of Volume You Are Using?
    ->MT5A
    - Erase Records After Moving?>
    -       <Yes or No>
    ->No
    -  16 Records Moved to TSDLWS
    ->
```

Example of RE-INITIALIZE Command

```
    ->I
    ->
    ->
```

3.4.5 ANALYZING DATA (FUNCTION - $\varepsilon\ \mu\ \sigma$)

The analysis of data is achieved through the F(FUNCTION) command in "Retrieve and Manipulate Data".

F(FUNCTION) The following functions are available to the user with this command.
1) Total column data
2) Mean & Variance (Standard Deviation)
These functions are performed on a column.

Example of FUNCTION (TOTAL) Command:

```
- Ready for Manipulation>
->
->F
-
- The Following Functions Are Available:
-      1) Total Column Data
-      2) Mean & Variance
-<<Enter Selection By Number?>
->1
- Column to Use?>
->1
-
-  Column 1 Totals =     136
-  With        16 Active Records
->
```

Example of FUNCTION (MEAN & VARIANCE) Command:

```
->F
-
- The Following Functions Are Available:
-     1) Total Column Data
-     2) Mean & Variance
- <<Enter Selection by Number?>
->2
- Column To Use?>
->1
-
- Mean & Variance of Column 1 For:
-
-         Records =          16
-         Mean    =          8.50
-         STD.DEV.=          4.76
->
```

3.4.6 PRINTING A RELATION (PRINT)

The printing of a relation is achieved through the P(PRINT) command in Retrieve and Manipulate Data

P(PRINT) Prints relations' data according to user's specifications. The following options are available to the user. The user may print the relation to the

1) Console
2) Printer
   or
3) Plotter

Sometimes it is possible to print a relation without specifying the tab positions. However, more often than not the user will have to supply FEASIL with the tab positions of each column. There are 72 columns available to print on the console and 132 columns available for the printer. For example, the scheme for printing a relation could be as follows using the P(PRINT) command.

Example of PRINT Command:

```
- Ready For Manipulation>
->
->P
-
- <Select Output Device>
- <Options Are:  1 -Console
-                2 -Printer
-                3 -Plotter
- <Enter Selection By Number 1,2 OR 3>
->1
-
- Print How Many Columns?>
->2
- Print Column Number 1?>
->1
- Print Column Number 2?>
->2
-  Delimiter Between Columns<May Be Null>?>
->
-  Empty Data Delimiter<May Be Null>?>
->
-  Do You Want Columns Aligned?>
->Yes
-  Position to Begin Column  1?>
->20
-  Position to Begin Column  2?>
->40
-      ?Are Column Alignments OK?
-               <Yes or No>
->Yes
```

```
-   Do You Want Titles?>
-   <Yes or No>
->No
-                     SYS                 SYST
-                     1                   S
-                     2                   SP
-                     3                   SSP
-                     4                   TSCB
-                     5                   HWK
-                     6                   RDL
-                     7                   HEL
-                     8                   RKN
-                     9                   SEMI
-                     10                  PERSH
-                     11                  IHWK
-                     12                  UNK
-                     13                  RDEYE
-                     14                  SVM
-                     15                  PTRT
-                     16                  RADOME
- >>Output Complete>
->
->
```

3.4.7 TERMINATION OF RELATION MANIPULATION (QUIT)

Q(QUIT)   To get out of retrieve and manipulate one uses the Q(QUIT) command as follows:

Example of QUIT Command:

```
->Q
->
-
- Do You Wish to Continue in Database System?
- <Yes or No>
->
```

For more explanations and examples of other commands in retrieve and manipulate refer to [3].

3.5 EDITING A RELATION

3.5.1 GETTING INTO FEASIL EDIT MODE

After "marking on" the disk and "signing on" the following procedure is used to get into FEASIL edit mode.

```
*FEASIL
    - *°*°* You Are In DMS(VER F6.05).  Do You Want To:
    - 1) Create a New Relation
    - 2) Delete a Relation
    - 3) Edit a Relation
    - 4) Modify Column Specifications
    - 5) Merge Two Relations
    - 6) Reorganize a Relation
    - 7) Retrieve & Manipulate Data
    - 8) Backup a Relation
    - 9) Status a Database
    - Enter Selection By Number >
    ->3
    -
    - Name of Relation to Be Edited?>
    - >SDLWS
    - Is the System Volume Being Used?
    - <Yes or No>
    ->No
    - What is Name of Volume You Are Using?
    ->MT5A
    -
    - Relation SDLWS   Has:      16 Records    2 Columns.
    - Current Record:       1
    - Current Column:       1
    -
    - Display Column Names?<Yes or No>
    ->Yes
    -
    - Display How Many Columns?>
    ->2
    - Display Column Number 1?>
    ->1
    - Display Column Number 2?>
    ->2
    - Ready to Edit>
    - >
    ->
```

After the user has entered the FEASIL edit mode, the list of commands in Table 3.12 is available to edit relations.

COMMAND	FUNCTION
A(Add)	Adds data to a relation from mechanical sources such as tape, cards, disk file, etc. (For example see Section 3.3, Step 6.)
B(Bottom)	Inserts tuple at bottom of relation prompts user for column values
C(Column)	List names of each column by number in the relation
D[](Delete)	Deletes [X] number of tuples starting with current tuple. None deleted if none specified
E(Examine)	Displays to user contents of current row and column pointer location
F(Find)	Find the first occurance of data item in specified column
I(Insert)	Inserts row after current row pointer. Prompts user for column values.
P[](Print)	Prints [X] rows of a relation starting with current tuple. One assumed if 0 or blank
Q(Quit)	Terminates editing
R(Re-start)	User may restart edit process to change display format
S(Substitute)	User substitutes new data into location pointed to by current row and current column pointers
?(Status)	Gives current values of row and column pointers relation name, number of records and number of columns.
+[] (Plus)	Moves current row pointer forward [X] rows. Last row given if total exceeded and 1 is understood if [0] or [] used
-[] (Minus)	Same as + except pointer moves backward

Table 3.12 FEASIL Edit Commands

The following sections give examples and further explanations of the following commands.

1) Bottom
2) +/-
3) ?(Status)/Examine/Substitute
4) Insert/Delete
5) Print
6) Quit

3.5.2 ADDING DATA TO THE BOTTOM OF A RELATION

The command B(Bottom) may be used to add data to the bottom of a relation.

Example of BOTTOM Command:

```
- Ready to Edit>
->
->B
-   1.SYS >
->17
-   2.SYST>
- >
->COFAC
- SYS >    17
- SYST>COFAC
- >
->
```

3.5.3 MOVING FORWARD OR BACKWARDS IN A RELATION

The command +[X] (Plus) may be used to move forward (towards the end) X number in a relation of rows while the command - [X] (Minus) may be used to move backward (towards the beginning) X number of rows in a relation. These commands will change the current row printer. These commands will list the contents of the columns that were specified by the user when first entering FEASIL edit mode. The contents will be listed in a vertical fashion. If the user responds Yes to the question

Display Column Names?

then the column names will be listed to the left of the appropriate data.

Example of PLUS Command:

```
->+2
- SYS >     15
- SYST>PTRT
- >
```

Example of MINUS Command:

```
->-4
- SYS >      13
- SYST>RDEYE
- >
```

3.5.4 CHANGING THE CONTENTS OF A COLUMN

The following are steps to use when changing the contents of a column:

1) Use the command ?(Status) to find the current row and column pointer. Optional
2) Use the command E(Examine) to move the column pointer to the desired column and to look at the contents of that column.
3) Use the command S(Substitute) to change the contents of the location pointed to by the current row and current column pointer.

Example of STATUS, EXAMINE, and SUBSTITUTE Commands:

```
- Ready to Edit>
- >
->?
-
- Relation SDLWS    Has:        17 Records    2 Columns.
- Current Record:        1
- Current Column         1
- >
->E
-     Column?>
->2
- SYST>S
- >
->S
- Column:
- SYST>S
-     Enter New String:>
- >
->T
- SYS >       1
- SYST>T
- >
->
-
```

3.5.5 INSERTING OR DELETING DATA

To add a row in any position in the relation the I(Insert) command is used. The row will be inserted after the current row pointer. The command D[X] (Delete) removes X number rows from the relation. Deletion begins with the current row pointer. None are deleted if none are specified.

Example of INSERT Command:

```
->I
-   1.SYS >
->41
-   2.SYST>
- >
->DFGT
- SYS >        41
- SYST>DFGT
- >
->
```

3.5.6 LOOKING AT THE CONTENTS OF ROWS

The command P[X] (Print) may be used to look at the contents of [X] number of rows starting with the current row and column pointer. This is different from the P(Print) command in retrieve and manipulate because the *contents are displayed only to the console.*

Example of PRINT Command:

```
->P16
-SYS >1   SYST >S
-SYS >2   SYST >SP
-SYS >3   SYST >SSP
-SYS >4   SYST >TSCB
-SYS >5   SYST >HWK
-SYS >6   SYST >RDL
-SYS >7   SYST >HEL
-SYS >8   SYST >RKN
-SYS >9   SYST >SEMI
-SYS >10   SYST >PERSH
-SYS >11   SYST >IHWK
-SYS >12   SYST >UNK
-SYS >13   SYST >RDEYE
-SYS >14   SYST >SVM
-SYS >15   SYST >PTRT
-SYS >16   SYST >RADOME
- >
```

3.5.7 GETTING OUT OF EDIT MODE

The command Q(Quit) terminates editing.

Example of QUIT Command:

```
->Q
-
- Do You Wish To Continue In Database System?
- <Yes or No>
->
```

For more explanations and examples of other commands refer to [3].

4.0 CONCLUSIONS AND RECOMMENDATIONS

Recommendations for continued progress in the development of software analysis fall into two principal areas. The first involves upgrading the FEASIL data base management system while the second area concerns the basic approach to data base file interaction.

A gradual but steady evolution of FEASIL is suggested. Enhancements should be added at a rate of one group every one to three months. As much as possible, upward compatibility of files should be maintained. The process suggested here is intended to satisfy these general criteria.

(i) FEASIL 7.00 would be the upgrade to the Fortran 7 compiler. Additional changes are as follows:

1) Removal of assembly language routines (if at all possible),
2) Addition of a "short string" strategy for strings up to eight characters,
3) Inclusion of a "revision number" in the TDF to indicate which version of FEASIL created the relation,
4) Fortran 7 file structures, and
5) Slight changes to make the system friendlier to novices.

This version should be implemented as soon as possible, and should operate on the same files as the earlier versions (if possible).

(ii) FEASIL 7.01 would add display functions. It could be ready about a month after 7.00, and be fully upward compatible. The added features would include the following:

1) Plots - one variable as a function of "time," two-variable scatter plots (with no line connecting points), a histogram, two time functions on one set of axes, and so on;
2) Functions - linear MMSE fit to data, "time" autocorrelation, cross correlation between two columns;
3) Interface - allow formatted input from text and binary input.

All plots could be typewriter art. They could appear on the screen of the terminal, and be copied to the printer for a permanent record.

(iii) FEASIL 7.10 would be a major revision of the ADF file structure with only minor changes visible to the user. Changes would include the following:

1) Modify the ADF to a standard block of 64 bytes (56 or 60 characters) with multiblock character strings allowed; the new input routine would deliver the 64 bytes, but would save the 256 byte physical block in core (remembering which one) to avoid unnecessary disk read operations; this would improve the speed for sequential reads, as well as allowing a longer maximum string length; and

2) Add an "uprev" command to convert 6.0 and 7.0 files to the new 7.1 format (this is one reason 7.00 added the version number).

This version would create only 7.1 files, but would still support rev 7.0 and earlier files. Future versions should support only 7.1 but keep the uprev command. The structure indicated above for the ADF may not be recommended exactly if statistical analysis of existing data bases suggests that a different packing factor would be better.

The current overall strategy for the number and size of data base files has lead to a system consisting of limited usage of the true structure philosophy. The effectiveness of the various data bases as tools for analysis would be greatly enhanced if a closer adherence to the true structure were employed. This would produce a larger number of files but of a more manageable size for manipulation. Subsequently, the amount of time necessary to sort and arrange test data into a suitable format for analysis would be decreased resulting in an increase in productivity.

REFERENCES

[1] Georgia Institute of Technology Memorandum HSV-80-091, DATE: June 18, 1980, FROM: E. J. Eifner, SUBJECT: Documentation of IHFTTEST.

[2] Final Report, EES/GIT Project A-2217, "Improved HAWK Target Signature and Test Data Base," by W. M. O'Dowd, Jr., Contract No. DAAK40-78-D-0008

[3] Technical Report RD-80-11, "A Relational-Based Data Management System for Engineering and Scientific Application," Maurice M. Hallum, III, Systems Simulation and Development Directorate, U. S. Army Missile Laboratory, June 1980.

[4] Technical Report RD-81-16, "Addition of Data Feature for FEASIL," Maurice M. Hallum, III, Systems Evaluation, Systems Simulation and Development Directorate, U. S. Army Missile Laboratory, U. S. Army Missile Command, Redstone Arsenal, Alabama 35898, August 1981.

[5] P. E. OS/32 Operator Reference Manual, S29-S74-R07.

APPENDIX A

CDC to Tape Transfer
Listing

```
HOTDG,MT1,CM120000.
ACCT(PN=XXXXXXXX,PBC=XXXXXXXX,OP=X)
PW=XXXXXXXX.
FTN.
ATTACH,TAPE10,DQXXXPT,ID=DQXXXR,CY=1.
TAPEPN,U,L IEZUAH,0020312.
REQUEST,TAPE11,HY,S,IB.          002031.  CLEAN TAPE DRIVE
FILE,TAPE11,BT=K,RT=S,MBL=132,PH=1.
LDSET(FILES=TAPE11)
LGO.
UNLOAD(TAPE11)
0000000000000000000000
      PROGRAM MAIN(INPUT,OUTPUT,TAPE6=OUTPUT,TAPE10,TAPE11)
      DIMENSION INT(25),REAL(100)
C
C$$$ TAKE DATA FROM PERMANENT FILE AND PUT ON TAPE
C$$$ USING FORMATTED WRITE IN BCD TAPE FORMAT.
C
      I=0
50    READ(10) (INT(J),J=1,25),(REAL(K),K=1,100)
      IF(EOF(10) .NE. 0) GO TO 100
      IF((INT(2).LT.25).AND.(INT(3).LE.6).AND.(INT(4).EQ.81))GO TO 75
      IF(INT(1) .EQ. 0) GO TO 75
      IF(INT(10) .NE. 0) INT(10)=1
      WRITE(11,200)(INT(J),J=1,25),(REAL(K),K=1,100)
75    I=0
      GO TO 50
100   I=I+1
      IF(I .LT. 3) GO TO 50
      STOP
200   FORMAT(2(10(I10,1X)/),5(I10,1X),/,20(5(F20.5,2X)/))
      END
```

APPENDIX B

Tape to File
Listing

```
*%%% COPY 7 TRACK BCD TO @1 (FILENAME)
$WRITE TAKES DATA FROM 7 TRAK PUTS IT ON FILE @1.DAT
$IFNULL @1
$WRITE MUST SPECIFY FILENAME OF RECEIVING DAT FILE
$CLEAR
$ENDC
L COPY32
$BUILD C7T.CMD
INPUT BCD:,132,VAR
REW INPUT
COPY *,@1.DAT
END
$ENDB
ST ,CO=C7T.CMD,LOG=CON:,LIST=CON:
$IFG 0
$WRITE PROBLEM ON 7 TRACK FOLKS
$CLEAR
$ENDC
L FRFSS
ST
$IFG 0
$WRITE PROBLEMS IN FRFSS STUFF
$ENDC
$CLEAR
$EXIT
```

APPENDIX C

File to FEASIL Format
Listing

```
C
C%%% UAH L. .J. JAN., 1982
C%%% GET DATA FROM TAPE USING FORMATTED READ AND PUT IT
C%%% INTO FEASIL FORMAT.
C%%% LU3 IS ASSIGNED TO A FILE CONTAINING CONDITION CODES
C%%% IN A10 FORMAT.  FILENAME IS ENTERED FROM TERMINAL.
C%%% CCODE FILE IS CREATED USING THE EDITOR.
C
      INTEGER RN,IDATE,INBREC,ST
      CHARACTER*50 COM
      CHARACTER*20 IFIL,IFDF,INF
      CHARACTER*10 ICCODE
      DIMENSION REAL(100),INT(25),IF(5),IFF(5),INF1(5),
     *INT2(25),REAL2(100)
      OPEN(5,FILE='CON:')
      OPEN(4,FILE='PR:')
      OPEN(7,FILE='UAH1:TEMPF.DAT',RECL=132,STATUS='OLD')
C
C%%% GET FILENAME OF CONDITION CODE FILE
C
      WRITE(5,400)
      READ(5,500) (IF(I),I=1,5)
      DECODE(IF,600) IFIL
      OPEN(3,FILE=IFIL,STATUS='OLD')
C
C%%% GET FILENAME OF FINAL FEASIL FILE
C
      WRITE(5,450)
      READ(5,500) (IFF(I),I=1,5)
      DECODE(IFF,600) IFDF
      OPEN(6,FILE=IFDF,STATUS='NEW')
C
C%%% OPEN COMMUNICATIONS TO CDC DATA FILE FROM 7 TRACK
C
      WRITE(5,460)
      READ(5,500) (INF1(I),I=1,5)
      DECODE(INF1,600) INF
      OPEN(7,FILE=INF,RECL=132,STATUS='OLD')
C
C%%% ICOUNT IS THE NO. OF RECORDS WRITTEN IN FEASIL FORMAT AND
C%%% IAS IS THE DELIMITER BETWEEN COLUMNS REQUIRED BY FEASIL
C
      REWIND 7
      REWIND 6
      IAS='*'
      ICOUNT=0
      GO TO 02
C
C%%% READ RUNS AND CALCULATE DATES FOR TESTING
C
      READ(7,2000) (INT(J),J=1,25),(REAL(I),I=1,100)
40    READ(7,2000,END=1750)(INT2(J),J=1,25),(REAL2(I),I=1,100)
      FLAG=0
      WRITE(5,850) (INT(J),J=1,4)
      IDATE=INT(4)*10000+INT(3)*100+INT(2)
      IDATE2=INT2(4)*10000+INT2(3)*100+INT2(2)
C
C%%% TESTS FOR GOOD OR BAD RUNS
C
      IF(IDATE .NE. IDATE2) GO TO 10
      IF(INT2(1) .GT. INT(1)) GO TO 10
      IF(INT2(1) .EQ. INT(1)) GO TO 45
C
C%%% CALCULATE NO. OF BAD RUNS IN NEW FILE AND BACKSPACE OVER THEM
C
      INBREC=(INT(1)-INT2(1))*34
      WRITE(3,700)INBREC
700   FORMAT(1X,'INBREC = ',I5)
      BACKSPACE(3,IOSTAT=ST,ERR=1250,COUNT=INBREC)
      WRITE(3,1500)ST
1500  FORMAT(1X,'IOSTAT = ',I3)
C
C%%% ALL DO LOOPS OF THIS FORM REPOSITION A RUN NO. FOR RETESTING
C
      DO 82 JJ=1,25
      INT(JJ)=INT2(JJ)
82    CONTINUE
      DO 92 NN=1,100
      REAL(NN)=REAL2(NN)
92    CONTINUE
250   FORMAT(1X,4(I10,1X))
      GO TO 40
C
C%%% WRITE RUNS TO NEW FILE
C
10    WRITE(3,2000) (INT(J),J=1,25),(REAL(I),I=1,100)
      DO 83 JJ=1,25
      INT(JJ)=INT2(JJ)
83    CONTINUE
      DO 93 NN=1,100
      REAL(NN)=REAL2(NN)
93    CONTINUE
      GO TO 40

1250  WRITE(5,1000)
1000  FORMAT('ERROR IN BACKSPACE')
      GO TO 5000
C
C%%% WRITE LAST RUN TO NEW FILE
C
1750  WRITE(3,2000) (INT(J),J=1,25),(REAL(I),I=1,100)
02    REWIND 3
      FLAG=1
C
C%%% READ CDC DATA FILE AND CCODE FILE IN ORDER TO MERGE THEM
C
```

```
      READ(7,1000) (INT(J),J=1,25),(REAL(I),I=1,100)
      DO 300 KK=1,10000
   46 READ(8,2000,END=4000) (INT2(J),J=1,25),(REAL2(I),I=1,100)
      READ(9,700,END=6000) RN,ICCODE,COM,NDATE
   47 IDATE=INT(4)*10000+INT(3)*100+INT(2)
      IDATE2=INT2(4)*10000+INT2(3)*100+INT2(2)
      IF(IDATE2.LT.810625) GO TO 950
C
C%%% MERGE TWO FILES BY RUN NO. AND DATE
C
      IF(RN-INT(1)) 900,49,900
 49   IF(NDATE .NE. IDATE) GO TO 900
      IF (IDATE .GT. 810702) GO TO 03
      YONE=ABS(REAL(70))
      YTWO=ABS(REAL(71))
      IF(YONE .GT. YTWO) GO TO 1001
C
C%%% WRITE SPECIFIED DATA FROM TWO FILES INTO FEASIL FORMAT
C
      WRITE(6,3000)IDATE,IAS,INT(1),IAS,ICCODE,IAS,REAL(4),IAS
     $,REAL(5),IAS
     1,REAL(6),IAS,REAL(43),IAS,YONE,IAS
     8,COM,IAS
      GO TO 01
 1001 WRITE(6,3000)IDATE,IAS,INT(1),IAS,ICCODE,IAS,REAL(4),IAS
     $,REAL(5),IAS
     1,REAL(6),IAS,REAL(43),IAS,YTWO,IAS
     2,COM,IAS
 01   ICOUNT=ICOUNT+1
      DO 60 JJ=1,25
      INT(JJ)=INT2(JJ)
   60 CONTINUE
      DO 70 NN=1,100
      REAL(NN)=REAL2(NN)
   70 CONTINUE
 300  CONTINUE
C
C%%%
C
 400  FORMAT(' ENTER FILENAME, CONTAINING CONDITION CODES')
 450  FORMAT(' ENTER FILENAME, WHICH FINAL FEASIL FILE IS TO BE STOR
 460  FORMAT(' ENTER FILENAME OF CDC DATA FILE')
 500  FORMAT(5A4)
 600  FORMAT(A20)
 700  FORMAT(I5,A10,A50,I6)
 2000 FORMAT(2(10(I10,1X)/),5(I10,1X),/,20(5(F20.5,2X)/))
 3000 FORMAT(I6,A1,I10,A1,A10,A1,F20.5,30X,A1,/
     $,F20.5,50X,A1,/
     1,F20.5,A1,F20.5,A1,F20.5,17X,A1,/
     8,A50,20X,A1)
```

```
C
C%%% WRITE SPECIFIED DATA FROM CDC FILE AND CCODE FILE INTO FRASIL FO
C
[illegible]   YTHREE=ABS(REAL(9))
      YFOUR=ABS(REAL(13))
      IF (YTHREE .GT. YFOUR) GO TO 1002
      WRITE(5,3000)IDATE,IAS,INT(1),IAS,ICCODE,IAS,REAL(23),IAS
     $,REAL(29),IAS
     1,REAL(30),IAS,REAL(7),IAS,YTHREE,IAS
     2,COM,IAS
      GO TO 31
1002  WRITE(5,3000)IDATE,IAS,INT(1),IAS,ICCODE,IAS,REAL(23),IAS
     $,REAL(29),IAS
     1,REAL(30),IAS,REAL(7),IAS,YFOUR,IAS
     2,COM,IAS
      GO TO 31
C
C%%%  END-OF-FILE MESSAGES
C
4000  READ(9,700,END=4500) ICCODE
      WRITE(5,4250)
4250  FORMAT(' POSSIBLE ERROR: MORE CC S THAN TAPE DATA')
4500  WRITE(5,5000) ICOUNT
5000  FORMAT(' END OF FILE ENCOUNTERED ON DATA INPUT',/,
     *1X,I4,' TUPLES (ROWS) DEVELOPED')
      GO TO 3000
6000  WRITE(5,7000)
7000  FORMAT(' ERROR: MORE DATA THAN C CODES')
      GO TO 3000
C
C%%% RUN NO. IN CCODE FILE DOES NOT MATCH RUN NO. IN CDC FILE
C%%% PRINT ERROR MESSAGE AND READ CDC FILE AGAIN
C
900   WRITE(5,9000)(INT(LL),LL=1,4)
9000  FORMAT(' ERROR:RUN NUMBER MISMATCH',4(1X,I10))
   45 CONTINUE
      DO 80 JJ=1,25
      INT(JJ)=INT2(JJ)
   80 CONTINUE
      DO 90 NN=1,100
      REAL(NN)=REAL2(NN)
   90 CONTINUE
      IF(FLAG .EQ. 0) GO TO 40
      READ(5,2000,END=4000) (INT2(J),J=1,25),(REAL2(I),I=1,100)
      GO TO 47
```

Copy available to DTIC does not permit fully legible reproduction

```
  950 WRITE(3,9009)IDATE
 9009 FORMAT(' ERROR:WRONG TEST DATE----',I5)
      GO TO 45
C
C*** CLOSE LOGICAL UNITS
C
 5000 CLOSE(4)
      CLOSE(3)
      CLOSE(5)
      CLOSE(6)
      CLOSE(7)
      CLOSE(8)
      STOP
      END
```

DISTRIBUTION

DRSMI-R	1
DRSMI-LP, Mr. Voigt	1
DRSMI-RPR	15
DRSMI-RPT	1
DRSMI-RD	5